

## PLP - Primer Parcial - Verano 2019

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R), y se promociona con tres ejercicios bien menos (B-) y las respuestas adecuadas a las preguntas teóricas. Las notas para cada ejercicio son: -, I, R, B-, B. Poner nombre, apellido, número de orden y cantidad de hojas en la primera hoja, y numerar las hojas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras. Entregar cada ejercicio en hojas separadas.

### Ejercicio 1 - Programación funcional

En este ejercicio **no** se permite el uso de recursión explícita, a menos que se indique lo contrario. Para resolver un ítem se pueden utilizar las funciones definidas en los ítems anteriores, sin importar si fueron resueltos correctamente o no.

Considere la siguiente representación de *árboles estrictamente binarios* (cada nodo tiene 0 ó 2 hijos):

`data AEB a = Hoja a | Bin (AEB a) a (AEB a)`

- 1) Dar el tipo y el esquema de **recursión primitiva**. En este inciso está permitido utilizar recursión explícita.
  - ii) Dar el tipo y definir `foldAEB`, el esquema de **recursión estructural** para árboles estrictamente binarios.
- b) Un árbol binario de búsqueda es un árbol en el cual cada nodo es mayor o igual que todos los nodos en el subárbol izquierdo, y menor que todos los del subárbol derecho. De este modo, la operación de búsqueda de un elemento es más eficiente que en el caso de un árbol sin esta propiedad.

`type SearchTree a = AEB a`

Definir la función `elemSearchTree :: Ord a => a -> SearchTree a -> Bool` que decide si un elemento está o no en el árbol aprovechando la propiedad de ser un árbol de búsqueda.

- c) Dar el tipo y definir `podarHastaNivel`, que a partir de un árbol y un entero, devuelve el árbol pasado por parámetro pero con los primeros  $n$  niveles. **Hint:** decidir primero sobre quién hacer la recursión y luego dar el tipo.

### Ejercicio 2 - Cálculo Lambda Tipado

Se desea extender el lenguaje de cálculo lambda tipado para soportar vectores de  $k$  posiciones. El conjunto de términos y tipos se modifica de la siguiente manera:

$\sigma ::= \dots \mid \vec{\sigma} \quad M ::= \dots \mid (M_1, \dots, M_n) \mid \forall x \in M \text{ check } P$

Donde  $(M_1, \dots, M_n)$  es el constructor para vectores que tienen al menos un elemento, y sus elementos tienen el mismo tipo. Además,  $\forall x \in M \text{ check } P$  es un término que indica si al reemplazar las apariciones libres de  $x$  en  $P$  por cada elemento de  $M$ , se cumple la propiedad  $P$ . La extensión tiene que ser eficiente, por lo que se pide que:

- $\forall x \in (0, \underline{1}, \underline{2}) \text{ check isZero}(x)$  reduce en varios pasos a `false`, sin chequear si  $\underline{2}$  es efectivamente cero.
- $\forall x \in (0, 0, 0) \text{ check isZero}(x)$  reduce en varios pasos a `true`.

- a) Introducir las reglas de tipado para la extensión propuesta.
- b) Definir el conjunto de valores e introducir la/las regla/reglas de semántica *one-step* para la extensión propuesta.

### Ejercicio 3 - Inferencia de tipos

Dada la siguiente extensión al conjunto de términos para el cálculo  $\lambda$  con listas:  $M ::= \dots \mid \text{map}_{\sigma,\tau} \mid \text{foldr}_{\sigma,\tau}$

La modificación al sistema de tipos es la introducción de dos axioma de tipado para  $\text{map}_{\sigma,\tau}$  y  $\text{foldr}_{\sigma,\tau}$ :

$\mathbb{W}(\text{map}) = \emptyset \triangleright \text{map}_{a,b} : (a \rightarrow b) \rightarrow [a] \rightarrow [b]$  siendo  $a$  y  $b$  variables de tipo frescas.

$\mathbb{W}(\text{foldr}) = \emptyset \triangleright \text{foldr}_{a,b} : (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow [a] \rightarrow b$  siendo  $a$  y  $b$  variables de tipo frescas.

Se asumen dadas la extensiones correspondientes para  $\text{Erase}$  y  $\text{mgu}$ . Usar el algoritmo  $\mathbb{W}$  con esta nueva extensión para tipar las siguientes expresiones:

a.  $\text{foldr map}$

b.  $\text{foldr foldr}$

En caso de que alguna expresión no tipe, justifique en términos del algoritmo.

### Preguntas Teóricas

a. Decir si las siguientes afirmaciones son válidas. Justificar su respuesta.

i) Existe  $M \in \lambda^{bn}$  tal que  $\emptyset \vdash M : \sigma$  es derivable,  $M \rightarrow V$  y  $\emptyset \vdash V : \tau$  es derivable para algún  $\tau \neq \sigma$ .

ii) Para todo  $M \in \lambda^{bn}$ , si  $\emptyset \vdash \text{fix } M : \sigma$  es derivable, entonces existe  $V$  tal que  $\text{fix } M \rightarrow V$ .

b. Mostrar con un ejemplo que al modificar el algoritmo de inferencia de tipos para el caso **if-then-else** tomando  $S$  como  $S = \text{MGU}\{\sigma \doteq \tau, \rho \doteq \text{Bool}\}$ , entonces el algoritmo que se obtiene no es correcto.

c. Mostrar, si es posible, un término  $M$  tal que se cumpla:

- $\emptyset \vdash M : \sigma$  y
- $\mathbb{W}(\text{Erase}(M)) = \Gamma \vdash M' : \rho$  y
- $\sigma = \rho$  y
- $M \neq M'$ .

1	2	3	N
B	R	B	A

Corrigió: Franco

1) a) I) Siendo dada  $AEB a = Hoja a \mid Bin (AEB a) a (AEB a)$  tenemos que el esquema de recursión primitivo es de la forma:

$$Fold AEB r1 :: (a \rightarrow b) \rightarrow (AEB a \rightarrow a \rightarrow AEB a \rightarrow b \rightarrow b \rightarrow b) \rightarrow AEB a \rightarrow b \quad \checkmark$$

$$Fold AEB r1 FHoja FBin (Hoja x) = FHoja x \quad \checkmark$$

$$Fold AEB r1 FHoja FBin (Bin L x r) = FBin L x r (Fold AEB r1 FHoja FBin L) (Fold AEB r1 FHoja FBin r) \quad \checkmark$$

Notar como en este esquema la función del paso recursivo toma tanto los hijos del nodo como los resultados de sus llamados recursivos.

$$II) Fold AEB :: (a \rightarrow b) \rightarrow (a \rightarrow b \rightarrow b \rightarrow b) \rightarrow AEB a \rightarrow b$$

$$Fold AEB FHoja FBin (Hoja x) = FHoja x$$

$$Fold AEB FHoja FBin (Bin L x r) = FBin x (Fold AEB FHoja FBin L) (Fold AEB FHoja FBin r)$$



En este inciso no se podía usar recursión explícita.

b) Considerando el tipo de árbol estrictamente binario en el que se cumple además la propiedad de ser binario de búsqueda, vemos que podemos

definir la función de búsqueda de un elemento aprovechando el esquema de recursión estructural y el hecho de que los nodos están ordenados:

$$elem SearchTree :: Ord a \Rightarrow a \rightarrow SearchTree a \rightarrow Bool$$

$$elem SearchTree elem = fold AEB (== elem) (\x L r \rightarrow x == elem \mid (if elem < x then L else r)) \quad \checkmark$$

c) Considerando que para poder el árbol necesitamos iterar tanto sobre el árbol como por la cantidad de niveles  $n$  poder, podemos aprovechar el esquema

de recursión estructural para formar una función que al evaluarse sobre  $n$  permite poder el árbol. Consideramos que  $n > 0$  en la entrada.

$$poder Hasta Nivel :: AEB a \rightarrow Int \rightarrow AEB a \quad \checkmark$$

$$poder Hasta Nivel = fold AEB (\x \rightarrow \backslash n \rightarrow (Hoja x)) (\x FL Fr \rightarrow \backslash n \rightarrow if n == 1 then (Hoja x) else Bin ((FL (n-1)) x (Fr (n-1)))) \quad \checkmark$$

(Quedaría más claro usando cláusulas where para las funciones que son parámetro de fold AEB).

Como se puede ver, tanto en el caso base como en el paso recursivo del esquema de recursión estructural se retorna una función que aplicada

sobre  $n$  permite poder el árbol. Se ve como para todo  $n$  que llegue a la hoja esto se construye por ser  $n > 1$  a la entrada y en Bin por ser  $n = 1$ .



2) a) Se deben tipar los términos  $(M_1, \dots, M_n)$  y  $\forall x \in M$  check P. Hacemos una regla de tipado en cada uso:

$$\mu(M_1, \dots, M_n): \frac{\text{Para todo } i \in 1..n, \Gamma \vdash M_i : \sigma \quad n > 0}{\Gamma \vdash (M_1, \dots, M_n) : \vec{\sigma}} \quad (\Gamma\text{-VECTOR})$$

$$\frac{* \forall x \in M \text{ check } p: \quad \Gamma \vdash M : \sigma \quad \Gamma \cup \{x : \sigma\} \vdash p : \text{Bool}}{\Gamma \vdash \forall x \in M \text{ check } p : \text{Bool}} \quad (\Gamma\text{-check})$$

b) En cuanto al conjunto de valores podemos tomar que los vectores  $e_i$  si pueden ser devueltos por un programa, por lo que  $V ::= \lambda (V_1, \dots, V_n)$ .

Con tenermos las reglas de semejanzas:

\*  $M_j \rightarrow M'_j \text{ if } j \neq n$  (E-Vector) \*  $M \rightarrow M'$  (E-Check1)  
 $(V_1, \dots, V_j, M_j, M_{j+1}, \dots, M_n) \rightarrow (V_1, \dots, V_{j-1}, M'_j, M_{j+1}, \dots, M_n)$   $\forall x \in M \text{ check } P \rightarrow \forall x \in M' \text{ check } P$   
 \*  $P \rightarrow P'$  (E-Check2) \*  $P \{x \leftarrow v_i\} \rightarrow \text{False}$  (E-Check False)  
 $\forall x \in V \text{ check } P \rightarrow \forall x \in V \text{ check } P'$   $\forall x \in (V_1, \dots, V_n) \text{ check } P \rightarrow \text{False}$  ESTD No CO small step

\*  $P\{x \leftarrow V_1\} \rightarrow \text{true} \quad n \geq 1$  (E-CHECK TRUE N) \*  $P\{x \leftarrow V_1\} \rightarrow \text{true}$  (E-CHECK TRUE SINGLE)  
 $\forall x \in (V_1, \dots, V_n) \text{ check } P \rightarrow \forall x \in (V_1, \dots, V_n) \text{ check } P$   $\forall x \in (V_1) \text{ check } P \rightarrow \text{true}$

Téase como en la nomenclatura se reduce a ~~False~~ False apenas la sustitución de  $x$  por el primer término del vector en  $P$  se reduce a False

No es necesario reducir todo el vector  
~~Alban~~ pero está ok.

~~P~~ treue  
Var. libres  
→ No puede  
reducir.

3) Considerando que  $\mathcal{W}(\text{map}) = \emptyset \triangleright \text{map}_{a,b} : (a \rightarrow b) \rightarrow [a] \rightarrow [b]$  con  $a, b$  variables frescas, y:

$\mathcal{W}(\text{foldr}) = \emptyset \triangleright \text{foldr}_{a,b} : (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow [a] \rightarrow b$  con  $a, b$  variables frescas, tenemos:

a. foldr map <sup>(3)</sup>

/ \  
foldr map  
(1) (2)

(1)  $\mathcal{W}(\text{foldr}) = \emptyset \triangleright \text{foldr}_{t_1, t_2} : (t_1 \rightarrow t_2 \rightarrow t_2) \rightarrow t_2 \rightarrow [t_1] \rightarrow t_2$

(2)  $\mathcal{W}(\text{map}) = \emptyset \triangleright \text{map}_{t_3, t_4} : (t_3 \rightarrow t_4) \rightarrow [t_3] \rightarrow [t_4]$

(3)  $S = \text{MGU} \{ (t_1 \rightarrow t_2 \rightarrow t_2) \rightarrow t_2 \rightarrow [t_1] \rightarrow t_2 \doteq ((t_3 \rightarrow t_4) \rightarrow [t_3] \rightarrow [t_4]) \rightarrow t_5 \} \xrightarrow{1} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4) \rightarrow [t_3] \rightarrow [t_4], t_2 \rightarrow [t_1] \rightarrow t_2 \doteq t_5 \} \xrightarrow{2} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4) \rightarrow [t_3] \rightarrow [t_4], t_5 \doteq t_2 \rightarrow [t_1] \rightarrow t_2 \} \xrightarrow{3} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4) \rightarrow [t_3] \rightarrow [t_4], t_5 \doteq t_2 \rightarrow [t_1] \rightarrow t_2 \} \xrightarrow{4} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4) \rightarrow [t_3] \rightarrow [t_4], t_5 \doteq t_2 \rightarrow [t_1] \rightarrow t_2 \} \xrightarrow{5} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4) \rightarrow [t_3] \rightarrow [t_4], t_5 \doteq t_2 \rightarrow [t_1] \rightarrow t_2 \} \xrightarrow{6} \emptyset$

luego  $S = \{ t_3 \leftarrow t_4, t_2 \leftarrow [t_1], t_1 \leftarrow t_4 \rightarrow t_4, t_5 \leftarrow [t_4] \rightarrow [t_4 \rightarrow t_4] \rightarrow [t_4] \}$  y con ello:

$\mathcal{W}(\text{foldr map}) = \emptyset \triangleright \text{foldr}_{t_4 \rightarrow t_4, [t_4]} \text{map}_{t_4, t_4} : [t_4] \rightarrow [t_4 \rightarrow t_4] \rightarrow [t_4]$

b. foldr foldr <sup>(3)</sup>

/ \  
foldr foldr  
(1) (2)

(1)  $\mathcal{W}(\text{foldr}) = \emptyset \triangleright \text{foldr}_{t_1, t_2} : (t_1 \rightarrow t_2 \rightarrow t_2) \rightarrow t_2 \rightarrow [t_1] \rightarrow t_2$

(2)  $\mathcal{W}(\text{foldr}) = \emptyset \triangleright \text{foldr}_{t_3, t_4} : (t_3 \rightarrow t_4 \rightarrow t_4) \rightarrow t_4 \rightarrow [t_3] \rightarrow t_4$

(3)  $S = \text{MGU} \{ (t_1 \rightarrow t_2 \rightarrow t_2) \rightarrow t_2 \rightarrow [t_1] \rightarrow t_2 \doteq ((t_3 \rightarrow t_4 \rightarrow t_4) \rightarrow t_4 \rightarrow [t_3] \rightarrow t_4) \rightarrow t_5 \} \xrightarrow{1} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4 \rightarrow t_4) \rightarrow t_4 \rightarrow [t_3] \rightarrow t_4, t_2 \rightarrow [t_1] \rightarrow t_2 \doteq t_5 \} \xrightarrow{2} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4 \rightarrow t_4) \rightarrow t_4 \rightarrow [t_3] \rightarrow t_4, t_2 \rightarrow [t_1] \rightarrow t_2 \doteq t_5 \} \xrightarrow{3} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4 \rightarrow t_4) \rightarrow t_4 \rightarrow [t_3] \rightarrow t_4, t_2 \rightarrow [t_1] \rightarrow t_2 \doteq t_5 \} \xrightarrow{4} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4 \rightarrow t_4) \rightarrow t_4 \rightarrow [t_3] \rightarrow t_4, t_2 \rightarrow [t_1] \rightarrow t_2 \doteq t_5 \} \xrightarrow{5} \{ t_1 \rightarrow t_2 \rightarrow t_2 \doteq (t_3 \rightarrow t_4 \rightarrow t_4) \rightarrow t_4 \rightarrow [t_3] \rightarrow t_4, t_2 \rightarrow [t_1] \rightarrow t_2 \doteq t_5 \} \xrightarrow{6} \text{Falla}$

donde 1: Renaming, 2: Swap, 3: Substitución de variables, 4: Occur Check, 5: Substitución de variables con listas. (MGU)

## Preguntas teóricas:

a. I) Dicho  $M \in \lambda^{b\eta}$  no existe ya que al cálculo lambda al extenderse con los naturales preservamos la propiedad de preservación, que dice que para toda reducción de un término cualquiera conado y bien tipado su tipo se conserva.

II) Falso, ya que al introducir el término  $\text{fix } M$  entre las expresiones del modelo  $\lambda^{b\eta}$  la propiedad de terminación ya no vale puesto que no se puede asegurar que las expresiones definidas recursivamente a partir de sub términos. Ej:

$$\begin{array}{l} x: \sigma \in \Gamma \quad \checkmark \\ \hline \Gamma = \{x: \sigma\} \Rightarrow x: \sigma \quad (\Gamma\text{-Abs}) \\ \hline \phi \vdash (\lambda x: \sigma. x): \sigma \rightarrow \sigma \quad (\Gamma\text{-Var}) \\ \hline \phi \vdash \text{fix}(\lambda x: \sigma. x): \sigma \quad (\Gamma\text{-Fix}) \end{array}$$

$\phi \vdash \text{fix}(\lambda x: \sigma. x)$  es derivable, pero:

$$\text{fix}(\lambda x: \sigma. x) \xrightarrow{(\Gamma\text{-Fix})} x \text{ s.t. } x \in \{ \text{fix}(\lambda x: \sigma. x) \} = \text{fix}(\lambda x: \sigma. x) \quad (\text{la reducción llega al mismo estado})$$

b. No el término  $\text{if } x \text{ then } x \text{ else } 0$  menos que:

$$\begin{array}{l} \text{(1)} \quad x: \text{Bool} \in \Gamma \quad \text{(2)} \quad x: \text{Nat} \in \Gamma \\ \hline \Gamma \vdash x: \text{Bool} \quad \Gamma \vdash x: \text{Nat} \quad \Gamma \vdash 0: \text{Nat} \\ \hline \Gamma \vdash \text{if } x \text{ then } x \text{ else } 0: \text{Nat} \end{array}$$

Dando (1) y (2) no pueden cumplirse a la vez pues al cambiar el juicio de tipo de  $x$  cada variable a un mismo tipo. Luego, siendo que por 0 el término solo podría tiparse a Nat, este término no es tipable. No obstante,

$$\text{if } x \text{ then } x \text{ else } 0 \quad \text{(1)} \quad \mathcal{W}(x) = \{x: t_1\} \Rightarrow x: t_1 \quad \text{(2)} \quad \mathcal{W}(x) = \{x: t_2\} \Rightarrow x: t_2 \quad \text{(3)} \quad \mathcal{W}(0) = \phi \Rightarrow 0: \text{Nat}$$

$$\begin{array}{ccc} x & x & 0 \\ \text{(1)} & \text{(2)} & \text{(3)} \end{array}$$

$$(4) \quad S = \text{MGU}(\{t_2 = \text{Nat}, t_1 = \text{Bool}\}) \xrightarrow{\text{Bool}/t_1} \{t_2 = \text{Nat}\} \xrightarrow{\text{Nat}/t_2} \phi \Rightarrow S = \{t_1 = \text{Bool}, t_2 = \text{Nat}\} \text{ y luego}$$

$$\mathcal{W}(\text{if } x \text{ then } x \text{ else } 0) = \underbrace{\{x: \text{Bool}, x: \text{Nat}\}}_{\text{inconsistencia}} \Rightarrow \text{if } x \text{ then } x \text{ else } 0: \text{Nat}$$

c. Sea  $M = (\lambda f: \text{Nat} \rightarrow \text{Nat}. \text{true}) (\lambda x: \text{Nat}. x)$  menos que:

$$\begin{array}{l} \{f: \text{Nat} \rightarrow \text{Nat}\} \Rightarrow \text{true}: \text{Bool} \quad (\Gamma\text{-True}) \\ \hline \phi \vdash \lambda f: \text{Nat} \rightarrow \text{Nat}. \text{true}: (\text{Nat} \rightarrow \text{Nat}) \rightarrow \text{Bool} \quad (\Gamma\text{-Abs}) \\ \hline \phi \vdash (\lambda f: \text{Nat} \rightarrow \text{Nat}. \text{true}) (\lambda x: \text{Nat}. x): \text{Bool} \quad (\Gamma\text{-App}) \end{array}$$

Q m res,  $\text{Eros}(M) = (\lambda f. \text{true}) (\lambda x. x)$  y con este:

$$\begin{array}{l} (\lambda f. \text{true}) (\lambda x. x) \quad \text{(1)} \quad \mathcal{W}(\text{true}) = \phi \Rightarrow \text{true}: \text{Bool} \quad \text{(2)} \quad \mathcal{W}(x) = \{x: t_1\} \Rightarrow x: t_1 \\ \hline \lambda f. \text{true} \quad \lambda x. x \quad \text{(3)} \quad \mathcal{W}(\lambda f. \text{true}) = \phi \Rightarrow \lambda f: t_2. \text{true}: t_2 \rightarrow \text{Bool} \text{ siendo } t_2 = t_1 \\ \hline \text{true} \quad x \quad \text{(4)} \quad t_x = t_1 \Rightarrow \mathcal{W}(\lambda x. x) = \phi \Rightarrow \lambda x: t_1. x: t_1 \rightarrow t_1 \end{array}$$

$$(5) \quad S = \text{MGU}(\{t_2 \rightarrow \text{Bool} = (t_1 \rightarrow t_1) \rightarrow t_3\}) \xrightarrow{t_1=t_1/t_2} \{t_2 = t_1 \rightarrow t_1, \text{Bool} = t_3\} \xrightarrow{\text{Bool}/t_3} \{t_2 = t_1 \rightarrow t_1\} \xrightarrow{\phi} \phi$$

Luego,  $S = \{t_2 \leftarrow t_1 \rightarrow t_1, t_3 \leftarrow \text{Bool}\}$  y con ello,  $\mathcal{W}((\lambda f. \text{true}) (\lambda x. x)) = \underbrace{\phi}_{\Gamma} \vdash \underbrace{(\lambda f. t_1 \rightarrow t_1. \text{true}) (\lambda x: t_1. x)}_{M'}: \underbrace{\text{Bool}}_{\rho}$

Como se puede ver  $\rho = \sigma$  y si hacemos una sustitución de  $M'$  tal que  $t_1 \neq \text{Nat}$  llegamos a que  $M \neq M'$